Приватний заклад вищої освіти
Одеський технологічний університет «ШАГ»
Кафедра інформаційних технологій та фундаментальної підготовки

Випускна кваліфікаційна роботи бакалавра

«ІНФОРМАЦІЙНА СИСТЕМА УЗГОДЖЕННЯ ВЗАЄМОДІЇ КОРИСТУВАЧІВ (ЗА ПРИКЛАДОМ СЕРВІСУ KABANCHIK) (BACKEND)»

на здобуття бакалаврського рівня вищої освіти
зі спеціальності «122 Комп'ютерні науки»

Виконавці проєкту

№	П.І.Б.	Група
1	Кравчук Д. В.	КН-П-191
2	Сарієв Г. В.	КН-П-191
3	Лоскутников І. С.	КН-П-191
4	Самбаєва К. С.	КН-Д-191
5	Денисов Д. Д.	КН-А-191

Автор звіту

_____ Кравчук Дмитро Вітольтович

Одеса – 2023

АНОТАЦІЯ

УДК 004.9
Д-30

Кравчук Д.В. Інформаційна система узгодження взаємодії користувачів: Автореферат випускної кваліфікаційної роботи на здобуття бакалаврського рівня вищої освіти зі спеціальності «122 Комп'ютерні науки». - Одеса: ОТУШ, 2023 - 13 с.

Причина проекту полягала в тому, щоб створити середовище, де фахівці з різних сфер можуть пропонувати свої послуги, а замовники можуть легко знайти найкращих фахівців для задоволення своїх потреб. Головна мета проекту - забезпечити простий та надійний спосіб знаходження та надання різноманітних послуг. Ця система ґрунтується на існуючій системі kabanchik.ua.

Основними функціями платформи є створення завдань, система рейтингу та відгуків, реєстрація та створення профілів виконавців і замовників; пошук, сортування та фільтрація послуг за різними критеріями. Особлива увага приділяється безпеці та конфіденційності даних користувачів. Система гарантує високий рівень шифрування та захисту персональних даних.

Ріст популярності онлайн-платформ для замовлення послуг, а також потреба в зручному та надійному сервісі для знаходження виконавців, зробили проект актуальним. Забезпечуючи широкий вибір виконавців і надаючи їм можливість просувати та залучати нових клієнтів, проект також має на меті сприяти розвитку ринку послуг.

Передові методи розробки програмного забезпечення є основою для розробки проекту. Гнучкі методи, як Agile, дозволяють команді забезпечують поетапний розвиток проекту. Також використовуються сучасні технології програмування та системи контролю версій для забезпечення якості та ефективності розробки.

Ми також застосували методологію CI/CD для прискорення розробки. Це дозволило нам мати непреривну інтеграцію, кращу продуктивність, швидші цикли випуску білдів і раннє виявлення дефектів.

ЗМІСТ

ВСТУП

Без швидкого розвитку технологій і цифрової сфери життя сучасної людини неможливо уявити. Для задоволення своїх потреб все більше людей віддають перевагу онлайн-сервісам і зручним інструментам. З цієї причини ми створили проект цифрової платформи, яка об’єднує користувачів і підрядників.

Наша команда вірить у потужну силу співпраці та взаємовигідних відносин. Ми розробили сервіс, який допомагає людям знайти надійних підрядників для різних видів роботи, починаючи від ремонту та будівництва, закінчуючи розробкою веб-сайтів і маркетинговою підтримкою. Наша платформа дозволяє користувачам розмістити свої завдання, а наші партнери-підрядники пропонуватимуть свої послуги за конкурентоспроможними цінами.

У документі розглядаються проблеми, з якими ми зіткнулися під час розробки програмного забезпечення, а також описується повний процес розробки.

Звіт складається з трьох розділів, які описують загальні характеристики системи, деталі створення серверної частини та результати розгортання та тестування системи. Висновки показують, що було досягнуто в цьому розділі проекту, а загальний висновок — це сукупність звітів усіх учасників проекту.

ТЕХНІЧНЕ ЗАВДАННЯ ДО ПРОЕКТУ

ІНФОРМАЦІЙНА СИСТЕМА УЗГОДЖЕННЯ ВЗАЄМОДІЇ КОРИСТУВАЧІВ (ЗА ПРИКЛАДОМ СЕРВІСУ KABANCHIK)

Основна функціональність:

Розміщення завдань, пошук виконавців, управління користувачами, платіжна система.

● Вимоги до бекенду:

- ○ Архітектура:
  - ■ Організація файлів: Clean architecture.
- ○ Технології:
  - ■ Мова програмування: C#.
  - ■ Фреймворк: ASP.NET WebApi.
  - ■ ORM: Entity Framework Core для роботи з базою даних PostgreSQL.
  - ■ Мапер: AutoMapper для перетворення даних між моделями.
  - ■ Валідація: FluentValidation для перевірки вхідних даних.
  - ■ Аутентифікація та авторизація: Використання Azure AD B2C для управління користувачами.
  - ■ Unit of Work та Repository Pattern: Для управління доступом до даних.
  - ■ Документація API: Використання Swagger для автоматичного генерування документації.
  - ■ Тестування: Використання Postman для тестування API.

● Вимоги до DevOps-інженера:

- ○ Контейнеризація: Використання Docker для контейнеризації додатка.
- ○ Хостинг: Розгортання на Azure Virtual Machine.
- ○ Система контролю версій: Використання Azure Repos для управління кодовою базою.
- ○ CI/CD інструменти: Використання Jenkins для автоматизації процесів CI/CD.
- ○ Моніторинг: Налаштування Prometheus і Grafana для моніторингу системи.
- ○ Проксі-сервер: Використання Nginx в якості проксі-сервера.
- ○ Планування та розгортання інфраструктури проекту.
- ○ Автоматизація процесів розгортання, моніторингу та масштабування системи.
- ○ Встановлення та налаштування інструментів для CI/CD.
- ○ Забезпечення безпеки та контролю якості проекту.

○ Управління конфігурацією та забезпечення надійності системи.

● Вимоги до дизайнера:

○ Проектування користувацького інтерфейсу та користувацького досвіду.
○ Використання інструментів для дизайну: Фігма, Ілюстратор, Фотошоп.

● Вимоги до фронтенду:

○ Фреймворк: Використання Angular для розробки фронтенд-частини сайту.
○ Інтегроване середовище розробки: Використання WebStorm для розробки на Angular.
○ Управління залежностями: Використання npm для управління пакетами.

Загальний опис проекту:

Опис функціональності:

Розміщення завдань:

● Створення нового завдання з вказанням категорії, опису та інших необхідних полів.
● Редагування та видалення завдання.
● Управління статусом завдання (відкрите, в роботі, завершене).

Пошук виконавців:

● Пошук виконавців за категоріями, навичками, локацією та іншими параметрами.
● Відображення списку доступних виконавців з основною інформацією та рейтингом.
● Фільтрація результатів пошуку та сортування за різними критеріями.

Оцінки та відгуки:

● Можливість оцінювати виконавців та залишати відгуки після завершення завдання.
● Відображення рейтингу та відгуків на профілях виконавців.
Цільова аудиторія:
● Визначення цільової аудиторії відповідно до виду завдань та послуг, пропонованих на сайті.
● Аналіз потреб і очікувань користувачів щодо функціональності та користувацького досвіду.

Розширюваність:

- Розробка модульної архітектури, що дозволяє легко додавати нові функції та можливості.
- Розгляд можливості інтеграції з іншими сервісами або платформами для розширення функціональності сайту.

Дизайн та користувацький інтерфейс:

Загальний стиль дизайну:

- Визначення бажаного стилю дизайну (мінімалістичний).
- Визначення колірної палітри, шрифтів та інших візуальних елементів, що відповідають обраному стилю.

Макети та прототипи:

- Створення макетів користувацького інтерфейсу для різних сторінок та функціональних елементів сайту.
- Розробка інтерактивних прототипів для демонстрації користувацької взаємодії та навігації по сайту.

База даних та зберігання даних:

Вибір та опис бази даних:

- Розробка структури бази даних, включаючи таблиці, зв'язки між ними та ключові поля.
- Врахування вимог до продуктивності та масштабованості бази даних.

Зберігання даних:

- Визначення даних, які повинні зберігатися в базі даних, включаючи інформацію про користувачів, завдання, коментарі тощо.
- Розробка моделей даних та визначення зв'язків між ними.

Безпека:

Аутентифікація та авторизація:

- Реалізація процесів аутентифікації користувачів з використанням Azure AD B2C.
- Встановлення прав доступу та ролей для різних типів користувачів.

Захист даних:

- Застосування заходів безпеки для захисту користувацьких даних, включаючи шифрування зберігання та передачі даних.

Інфраструктура:

Контейнеризація:

● Використання Docker для контейнеризації бекенд- та фронтенд-частини додатка.

● Створення Docker-контейнерів для різних компонентів додатка та їх оркестрація.

Хостинг:

● Розгортання додатка на Azure Virtual Machine або іншому облачному сервісі.

CI/CD:

● Використання Jenkins для налаштування процесів CI/CD.

● Автоматизація процесів розгортання, тестування та випуску нових версій додатка.

Моніторинг:

● Налаштування Prometheus та Grafana для моніторингу системи.

● Відстеження метрик продуктивності, завантаження та інших параметрів для забезпечення стабільності та швидкодії сайту.

Система контролю версій:

Використання системи контролю версій:

● Інтеграція проекту з Git для управління версіями та історією змін.

● Створення репозиторію Git та налагодження процесу спільної роботи розробників.

Гілкування та злиття:

- Визначення стратегії гілкування та злиття коду, наприклад, використання моделі Git Flow.
- Створення та керування різними гілками для розробки нових функцій, виправлення помилок та випуску стабільних версій.

Робота з командою розробників:

- Опис угод щодо найменування гілок, коментарів до комітів та інших правил для спільної роботи над проектом.
- Визначення процесу рецензування коду та схвалення змін.

Інтеграція з CI/CD:

- Використання інструментів CI/CD, таких як Jenkins, для автоматизації процесів збірки, тестування та розгортання додатка.
- Налаштування пайплайнів CI/CD, що включають етапи збірки, тестування, аналізу коду та розгортання на сервері.

Управління версіями та тегування:

- Опис правил для присвоєння версій релізам, виправленням помилок та іншим змінам.
- Використання тегів Git для маркування важливих точок в історії проекту, таких як релізи або майлстоуни.

Резервне копіювання:

- Розробка стратегії резервного копіювання репозиторію Git для забезпечення збереженості коду та історії змін.

Управління проектом:

Створення плану проекту:

● Визначення етапів розробки, термінів та завдань.

● Встановлення пріоритетів та розподіл ресурсів. Комунікація та співпраця:

● Встановлення засобів комунікації та співпраці між членами команди.

● Використання інструментів для спільного редагування та обміну документами.

Звітність:

- Вихідний код повинен оформлятися у відповідності до загальноприйнятих стандартів, на зразок стандартів GNU http://www.gnu.org/prep/standards/, у тому числі з додержанням вимог до коментування та документування коду.

- Зберігання вихідних кодів ПЗ повинно здійснюватись за допомогою системи контроля версій (вибір конкретної системи узгоджується групою проекту). Структура файлів та каталогів повинна відбивати архітектуру ПЗ, на зразок:
    - root
    - Backend
    - Web
- Підготовка звітів про стан проекту та результати роботи.
- Презентація результатів перед командою та замовником проекту.

РОЗДІЛ 1
ЗАГАЛЬНА ХАРАКТЕРИСТИКА СИСТЕМИ

Система була спроектована за допомогою клієнт-серверної розподіленої архітектури відповідно до технічного завдання (ТЗ). Систему можна умовно розділити на наступні архітектурні частини:
- Design
- Frontend
- Backend
- DevOps

«Design»

У частині «Design» представлені моделі та описи поведінки користувача. Модель містить інтерфейси, які дозволяють користувачам взаємодіяти з інформаційною системою та переходити між ними. Підходи розумної навігації, такі як підходи UI/UX, висновки колористики та ергономіка мобільних пристроїв, були використані при розробці інтерфейсів. Частина розміщена за адресою https://www.figma.com/file/GJ5selmuva3f6GiDoassR7/%D0%94%D0%B8%D0%BF%D0%BB%D0%BE%D0%BC?type=design&node-id=0-1&mode=design&t=4Urx8anUmJU2AFwo-0.

«Frontend»

Частина «Frontend» розробки була реалізована за допомогою фреймворку Angular, мовою програмування TypeScript, а також мови гіпертекстової розмітки HTML та каскадних таблиць стилів CSS.
Для кастомізації інтерфейсу використовується Material Design та Bootstrap.
Клієнтська частина реєстрація, авторизація та автентифікація реалізовані за допомогою бібліотеки Microsoft Authentication Library (MSAL) та використовують сервіс Azure AD B2C.
Для зв'язку клієнтської частини з «бекендом» по протоколу HTTP використовуються методи класса “HttpClient” в TypeScript.

«Backend»

Серверна частина «Backend» розроблена за допомогою технології платформи Microsoft ASP.NET Core та архітектурного стилю API - RESTful.
Підхід Clean Architecture, або чиста архітектура, використовувався для проектування програмного забезпечення. Цей підхід дозволяє структурувати систему навколо основних ідей і принципів, щоб забезпечити високу зрозумілість, підтримуваність і тестируваність коду.
C# є основною мовою програмування. Дані платформи та користувачів зберігалися за допомогою PostgreSQL. Після ретельної оцінки та дослідження проекту було прийнято рішення використовувати цю СУБД (систему управління базами даних), оскільки вона відповідає архітектурі та технологіям, які

використовуються в кожній частині проекту. Це дозволило використовувати технологію ORM (Object Relational Mapping) Entity Framework Core від Microsoft. Підтримка мови запитів LINQ (Language Integrated Query) є однією з найпопулярніших сучасних технологій для розробників програмного забезпечення для створення SQL-запитів до баз даних. Це робить розробку та взаємодію з базою даних значно простішою.
Для “мапінгу” одних моделей на інші використовується популярна бібліотека AutoMapper. Валідацію вхідних даних від клієнтської частини забезпечується бібліотекою FluentValidation.
Для імплементації реєстрація, авторизація та автентифікація було обрано хмарний сервіс Azure Active Directory B2C, який надає такі переваги: гнучкість та розширюваність управління ідентифікацією для зовнішніх користувачів, захист даних та безпеку, готові рішення для ідентифікації соціальних мереж, аналітика та звітність, швидке впровадження.
Доступ до даних у системі організовані двома популярними паттернами проектування: Unit of Work та Repository. Патерни допомагають покращити структуру та організацію доступу до даних у системі, полегшуючи розробку, тестування та підтримку.

«DevOps»

Під час розробки наш devops-спеціаліст проаналізував усі можливості і прийшов до висновку, що для реалізації CI/CD маємо застосувати Jenkins, а для графіків і статистики “деплоїв” - Grafana.

Доступ до готової користувальницької частини онлайн-платформи знаходиться за адресою: https://busybee.space.

РОЗДІЛ 2
РЕАЛІЗАЦІЯ СЕРВЕРНОЇ ЧАСТИНИ

Серверна частина складається з:

- REST-API з використанням платформи розробки ASP.NET Core від Microsoft;
- Бази даних: PostgreSQL;
- Мови програмування: C#;
- Середовища розробки: Rider;
- Використанні технології: Entity Framework Core, LINQ;
- Встановлені додаткові пакети: AutoMapper, FluentValidation, Microsoft.Identity.Web.

Перед початком розробки Web API додатку необхідно було визначити функціональні вимоги додатку, визначивши його основні компоненти та їхні зв'язки. Для створення нового проекту C# Web API було використано IDE Rider. Також було налаштовано залежності для використання EF Core, LINQ, Swagger, AutoMapper і FluentValidation. Потужні функції рефакторингу та інтегрована підтримка інструментів розробки Rider роблять його корисним для використання.

У процесі розробки було використано чисту архітектуру (Clean Architecture), що дозволяє розділити додаток на логічні рівні та забезпечити їх незалежність один від одного. Це покращує підтримку та розширюваність майбутнього додатку. Анемічна доменна модель (Anemic domain model) дозволяє зосередитися на бізнес-логіці та доменних об'єктах, відокремлюючи їх від технічних питань, таких як валідація та збереження даних.

Для управління та доступу до даних були використані патерни Unit of Work та Repository. Для того, щоб переконатися, що база даних залишається безпечною, Unit of Work відповідає за керування змінами та комітами в базі даних. Repository відповідає за доступ до даних, абстрагуючи реалізацію збереження та витягування даних з бази даних. Це дозволяє зосередитися на бізнес-логіці та розділити її від механізмів доступу до даних. На основі цього патерну ми імплементували фільтрацію, пагінацію та сортування щоби покращити ефективність пошуку, зменшити обсяг передачі даних, полегшити навігацію та забезпечити користувачам зручність у виборі та сортуванні даних. Ці функції є стандартними у багатьох додатках, що працюють з базами даних, і допомагають покращити взаємодію з даними. Фільтрація, пагінація та сортування імплементована на рівні як API так і БД. Під API я маю на увазі те що, фронтенд може додавати до GET запиту такі query-параметри: номер сторінки, розмір сторінки, поле по якому сортувати(приклад, id, title або інше), напрямок сортування (за зростанням або за спаданням) та рядок тексту, по якій шукається значення в полях моделі. Під БД я маю на увазі те що, бекенд будує

такий запит до БД (IQueryable), який враховує всі параметри, які прийшли з фронтенду.

EF Core використовувався для зберігання даних, що забезпечує розширену функціональність і просту роботу з базою даних. Починаючи з конфігурації підключення до бази даних, було важливо враховувати різні проблеми, такі як неправильні дані або невідповідності в структурі бази даних. Підхід Code First дозволив автоматизувати створення таблиць на основі моделей даних, що дозволило зосередитися на розробці функціональності. В якості СУБД (Система управління базами даних) ми використовуємо PostgreSQL.

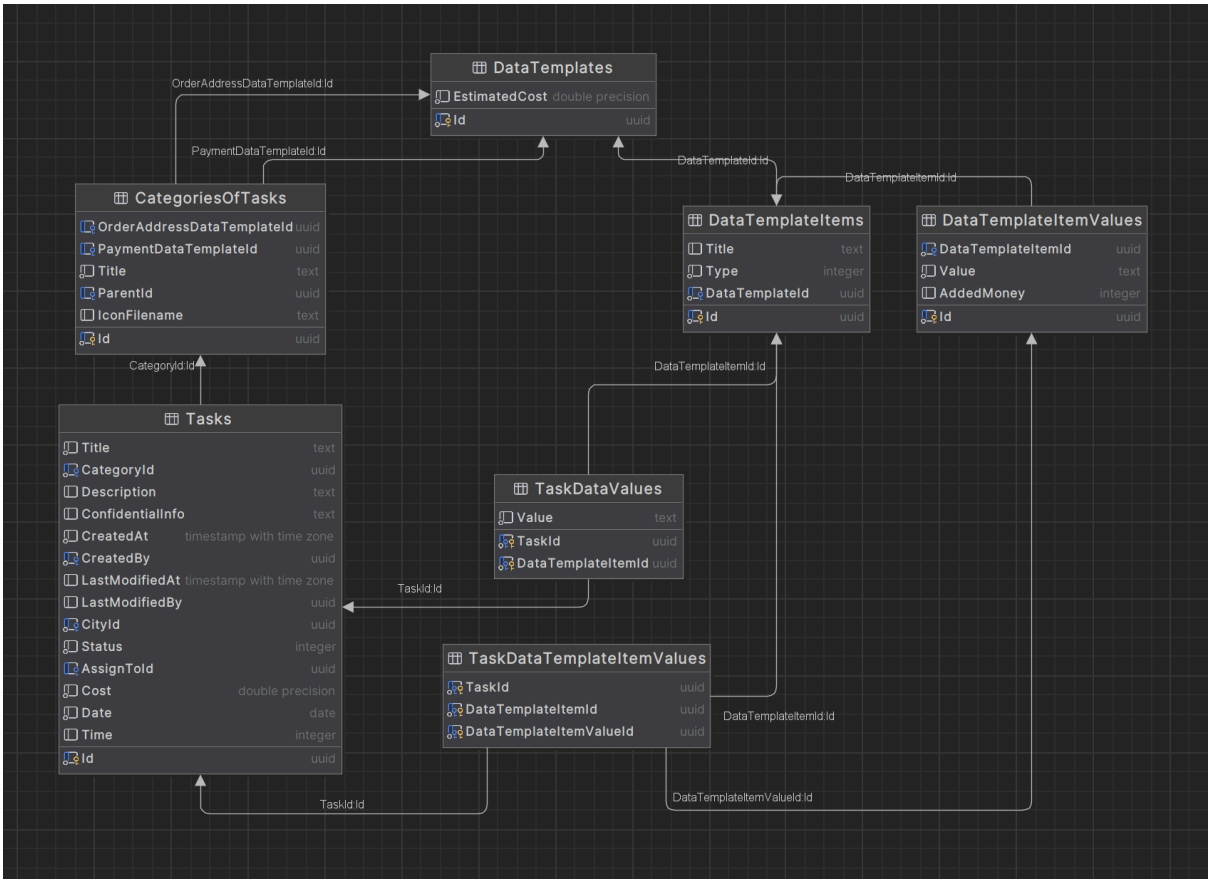
Розробка сервісів і контролерів є важливою частиною розробки Web API додатку. Контролери відповідають за обробку запитів і відправку відповідей, а сервіси виконують бізнес-логіку. Завдяки використанню LINQ взаємодія з базами даних стає простішою, що дозволяє виконувати запити, фільтрувати, сортувати та групувати дані. Це гарантує ефективну обробку запитів і швидкий доступ до необхідної інформації. Спочатку я зіткнувся з проблемою, пов’язаною з неефективним використанням LINQ-запитів, що призвело до повільного функціонування додатка. Тим не менш, використання вбудованих функцій EF Core, оптимізація запитів і створення індексів допомогли вирішити цю проблему.

Потужні інструменти для автоматичного маппінгу між моделями та валідації даних — AutoMapper і FluentValidation. Розширена перевірка дозволило встановлювати правила валідації для моделей, що гарантує їхню

коректність і цілісність. Це допомогло запобігти неправильним або неповним даним, які можуть спричинити неправильну роботу додатка. АвтоMapper полегшує процес маппінгу даних між об'єктами, забезпечуючи автоматичну конвертацію та зменшуючи кількість коду, що потрібно написати.

Azure AD B2C використовується для авторизації та захисту доступу до ресурсів додатка. Це дозволяє контролювати доступ користувачів до даних і функціональності. Налаштування Azure AD B2C вимагає створення та налаштування компонентів, таких як політика доступу та клієнти, що робить процес складним в порівнянні з іншими варіантами. Тим не менш, після успішної настройки Azure AD B2C пропонує надійний механізм авторизації.

Користувач після успішної реєстрації може створити нове завдання. Для цього потрібно перейти на сторінку проекту та заповнити всю необхідну інформацію. Завдання поділяються на різні категорії, що допомагає користувачам знайти відповідні послуги та виконавців для своїх потреб. Для створення завдання необхідно заповнити такі поля, як заголовок, опис, бюджет і термін виконання. Крім того, є унікальні поля, які можуть змінюватися залежно від типу завдання, яке ви вибрали. Наприклад, у розділі «Будівництво та ремонт» можуть бути додаткові поля, включаючи тип ремонту (наприклад, ремонт квартири, ремонт покрівлі тощо) і площа ремонту (наприклад, розмір приміщення або площа). Переді мною постала проблема як зберігати унікальні поля в базі даних для різних категорій, але при цьому мати можливість редагувати, видаляти та додавати унікальні поля до категорій без зміни структури БД.



Для цього я створив таблицю DataTemplates який має відношення до категорій (CategoriesOfTasks) - 1:N, тобто одному темплейту може відповідати 0, 1 або більше категорій. DataTemplate має набір DateTemplateItems, вони мають не обов'язкову назву, наприклад: Вантажники. Та мають тип поля, це може бути просто рядок, набір радіобатонів або набір чекбоксів. Якщо DateTemplateItems має тип поля - набір радіобатонів або набір чекбоксів. То для нього створюється сукупність DateTemplateItemValues, вони мають тільки обов'язкову назву та ціну яка додається до загального бюджету.

Для зберігання користувацьких файлів (аватарів та файлів-портфоліо) я використовую папку рядом з бекендом. Цей метод включає створення директорії на сервері, де знаходиться бекенд, а також збереження файлів користувачів безпосередньо в цій папці. Таке рішення було прийнято, оскільки об'єми файлів не такі великі та вони не сильно будуть рости. Таким чином, масштабування не буде проблемою для нас. Крім того, це рішення забезпечує контроль над безпекою користувацьких файлів, швидкий доступ до файлів і прямий доступ до них.

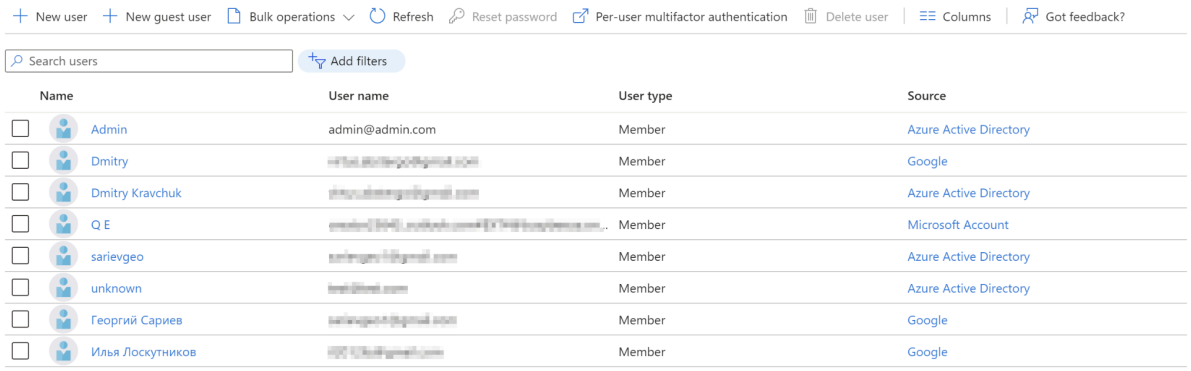
Для збереження та керування кодом був використаний репозиторій Azure DevOps. Це дозволяє керувати версіями коду, співпрацювати в команді та створювати централізоване сховище. Таким чином команда могла легко вносити зміни та відстежувати прогрес проекту. Тим не менш, коли кілька розробників працюють над одними й тими ж файлами, виникали проблеми з конфліктами злиття. Це вдалося уникнути за допомогою розуміння процесу злиття та використання правильних методів гілок.

РОЗДІЛ 3
АДМІНІСТРАТИВНА ЧАСТИНА ОНЛАЙН-ПЛАТФОРМИ

Платформа має два види користувачів: адміністратори та звичайні користувачі.

Адміністратор може редагувати, створювати та видаляти категорії, а також їх унікальні поля, про які писалося вище.

Azure AD B2C дозволяє адміністратору переглядати користувачів, які зареєструвалися на платформі, та адмініструвати їх включаючи управління їх реєстрацією, входом, профілями та іншими аспектами, пов'язаними з ідентифікацією.



ВИСНОВКИ

Під час розробки цього проекту було розглянуто та застосовано низку методів і технологій, щоб забезпечити ефективну взаємодію між компонентами клієнта та сервера додатку. Використання сучасних технологій, таких як REST API та ASP.NET Core, дозволило створити більш масштабовану систему та покращити продуктивність.

Особлива увага приділялася налаштуванню бази даних з використанням Entity Framework Core та LINQ, що дозволило оптимізувати запити та забезпечити швидкий доступ до даних. Успіх розробки залежав від знання цих інструментів і здатності вирішувати такі проблеми.

Azure AD B2C дозволив безпечно та захищено доступ до ресурсів додатку. Коли справа доходить до налаштування Azure AD B2C, виникали деякі труднощі, але використання цієї платформи допомогло успішно завершити авторизацію.

Azure DevOps відповідав за зберігання та керування кодом, що забезпечило ефективну командну роботу та керування версіями.

Загальною метою проекту було створення масштабованої та ефективної системи, яка включатиме API, авторизацію користувачів, засоби захисту даних, безпечне збереження та обробку даних і інші функції.

Після тестування та перевірки функціональності на різних пристроях було встановлено, що система працює належним чином і відповідає вимогам.

ПЕРЕЛІК ВИКОРИСТАНИХ ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ СЕРВЕРНОЇ ЧАСТИНИ

№ п.п.	**Ресурс**	**Ступінь запозичення**
1.	ASP.NET Core; Розробник: Microsoft; Документація: посилання.	Використано у якості платформи для розробки бекенд частини Web-додатку.
2.	Entity Framework Core; Розробник: Microsoft; Документація: посилання.	Використано як ORM-технологія для маніпуляції з даними, які зберігаються у базі даних.
3.	PostgreSQL; Розробник: PostgreSQL Global Development Group; Документація: посилання.	Використано як основна база даних
4.	Мова програмування C#; Розробник: Microsoft; Документація: посилання.	Основна мова програмування.
6.	Rider; Розробник: JetBrains; Документація: посилання.	Була використана у якості головного середовища розробки серверного програмного забезпечення.
8.	Декларативна мова запитів LINQ; Розробник: Microsoft; Документація: посилання.	Використовувалася для створення декларативних та оптимізованих запитів до бази даних.

13.	Бібліотека AutoMapper; Розробник: Jimmy Bogard; Документація: посилання.	Використано як об'єктно-об'єктний “мапер” для моделей.
14.	Бібліотека FluentValidation; Розробник: Jeremy Skinner; Документація: посилання.	Використано для валідації користувацьких даних що надійшли.
15.	Сервіс Azure AD B2C; Розробник: Microsoft; Документація: посилання.	Використано як платформу для реєстрація, авторизація та автентифікація.
16.	Swagger; Розробник: SmartBear Software; Документація: посилання.	Використано для тестування кінцевих точок API (endpoints).
17.	Сервіс Azure DevOps; Розробник: Microsoft; Документація: посилання.	Використано для зберігання коду в якості git-репозиторію.